Bab 4

Membuat dan Memanipulasi Tabel

Tabel adalah batu bata pembangun database. Sebuah database relasional tersusun dari tabel-tabel yang saling berhubungan satu sama lain. Itulah sebabnya mengapa jenis database ini dinamai *relasional*, sebab *relasi* berarti hubungan. Bab ini menerangkan cara membuat tabel, mengubah strukturnya, dan menghapusnya jika sudah tidak diperlukan.

Dalam mencoba bab ini, ada baiknya Anda melakukannya pada database kosong. Jika Anda memiliki akses ke user root MySQL, berikan perintah ini pada klien konsol mysql:

```
– katakanlah databasenya bernama bab4
CREATE DATABASE bab4;
USE bab4;
```

4.1 CREATE TABLE

```
-- Resep 4-1-1: Sintaks dasar untuk membuat tabel
CREATE TABLE nama_tabel (
     nama_kolom1 tipe_data [DEFAULT nilai_default] [constraint
 kolom...]
  [, nama_kolom2 tipe_data [DEFAULT nilai_default] [constraint
kolom...]
  [, ...]]
  [, constraint tabel...]
);
```

Perintah SQL untuk membuat tabel adalah CREATE TABLE. Perintah ini sering disebut juga sebagai DDL (data description/definition language), karena kalau seseorang ditanya seperti apa struktur tabel yang dimilikinya, maka dia bisa menjawab dengan memberikan perintah CREATE TABLE yang digunakan untuk membuat tabel tersebut. Jadi perintah ini bisa dipakai untuk merepresentasikan definisi struktur database.

Sintaks dasar perintah CREATE TABLE adalah seperti yang diberikan pada resep 4-1-1, yaitu definisi tiap kolom dipisahkan dengan koma beserta definisi constraint tabel (jika ada). Sebuah tabel minimal terdiri dari satu buah kolom. Setiap kolom memiliki tipe data dan boleh diberi nilai default dan satu atau lebih constraint kolom.

Seperti telah dijelaskan di Bab 3, nama tabel dan kolom merupakan identifier, dan diawali dengan huruf dan diikuti oleh huruf dan angka (atau underscore). Namun boleh saja nama tabel dan kolom mengandung karakter lain seperti spasi, asalkan selalu dikutip untuk mencegah ambiguitas.

Tipe-tipe data yang dikenal di SQL (dan khususnya MySQL) telah dijelaskan pada Bab 3 pula.

Apa itu constraint kolom? Constraint kolom adalah pembatasan atau pelarangan agar sebuah kolom tidak dapat berisi nilai-nilai tertentu. Contohnya adalah constraint NOT NULL, berguna agar kolom tidak boleh diisi nilai NULL. Kegunaan constraint kolom dan constraint-constraint kolom yang dikenal di SQL dijelaskan pada Subbab 4.2.

Constraint tabel pada dasarnya sama dengan constraint kolom, yaitu untuk membatasi agar kolom-kolom hanya dapat berisi nilai-nilai tertentu saja yang kita

inginkan. Perbedaannya, constraint tabel diterapkan pada sekelompok kolom sekaligus, sementara constraint kolom per kolom tunggal.

Berikut ini contoh sebuah perintah CREATE TABLE:

```
-- Contoh 4-1-1
CREATE TABLE buku (
  isbn CHAR(10) PRIMARY KEY,
  judul VARCHAR(150) NOT NULL,
  penerbit VARCHAR(64) NOT NULL,
  tglterbit DATE NOT NULL
);
```

Pada contoh di atas, tabel buku memiliki empat kolom: isbn, judul, penerbit, dan tglterbit. Kolom isbn, judul, penerbit bertipe data teks dengan panjang maksimum masing-masing 10, 150, dan 64 karakter. tglterbit bertipe data tanggal. Kolom isbn memiliki constraint PRIMARY KEY, sementara semua kolom lain juga memiliki constraint kolom NOT NULL. Semua kolom tidak didefinisikan memiliki nilai default, jadi kalau waktu diisi nilainya tidak disebutkan maka akan dipakai nilai default menurut tipe data masing-masing, misalnya '' (string kosong) untuk tipe data teks, 0 untuk bilangan, dan seterusnya.

4.2 Constraint

Seperti telah disinggung sebelumnya, constraint kolom (dan constraint tabel) berguna untuk membatasi sebuah kolom (atau kombinasi beberapa kolom) agar tidak bisa mengandung nilai tertentu, atau hanya dapat menyimpan nilai tertentu saja. Dikenal beberapa jenis constraint, di mana hanya sebagian saja yang didukung oleh MySQL.

NOT NULL constraint

```
-- Resep 4-2-1: NOT NULL constraint
CREATE TABLE ... (
  ...
  nama_kolom tipe_data [CONSTRAINT nama_constraint] NOT NULL
  ...
);
```

NOT NULL constraint berguna untuk melarang sebuah kolom agar tidak bisa menyimpan nilai NULL.

Constraint dapat diberi nama (dengan klausa CONSTRAINT *nama_constraint*). Jika tidak diberi nama, maka database akan memberi nama/identifier otomatis. Catatan untuk MySQL: di MySQL, constraint NOT NULL tidak bisa diberi nama. Jika sebuah kolom didefinisikan tanpa constraint ini, atau diberi "constraint" NULL (bukan NOT NULL), maka kolom dapat berisi NULL.

Pada umumnya, disarankan agar Anda selalu menambahkan constraint ini (karena alasan kecepatan dan untuk menjaga agar kolom selalu berisi nilai), kecuali jika kolom memang diperbolehkan menyimpan nilai yang tidak diketahui. Contohnya, form submisi guestbook atau registrasi di aplikasi web umumnya hanya mengandung beberapa field saja yang wajib diisi, sementara field-field lain boleh dikosongkan oleh pengunjung website. Dalam kasus ini, field-field yang tidak wajib diisi dapat diwakili dengan kolom-kolom di tabel yang boleh mengandung nilai NULL.

Kadang-kadang, terutama untuk kolom bertipe data string (CHAR/VARCHAR), Anda bisa mewakili informasi yang tidak diisi dengan string kosong. Sehingga kolomnya sendiri dapat Anda beri constraint NOT NULL. Tapi untuk kolom bertipe angka, lebih sering Anda perlu membedakan antara 0 dan "tidak diketahui." Dalam kasus ini, maka kolom harus boleh NULL.

Unique constraint

```
-- Resep 4-2-2: unique constraint
CREATE TABLE ... (
  ...
  -- sebagai constraint kolom
  nama_kolom tipe_data [CONSTRAINT nama_constraint] UNIQUE
  ...
  -- sebagai constraint tabel
  [CONSTRAINT nama_constraint] UNIQUE (nama_kolom[, nama_kolom2,
...])
  ...
);
```

Constraint ini berguna untuk membatasi agar sebuah kolom tidak mengandung dua baris yang memiliki nilai yang sama untuk kolom tersebut. Dengan kata lain, jika kolom integer bernama *a* didefinisikan UNIQUE, maka jika sebelumnya ada baris yang memiliki nilai *a*=1, maka jika kita memasukkan baris baru dengan *a*=1 juga, perintah INSERT tersebut akan gagal. Agar berhasil, baris dengan *a*=1 sebelumnya harus dihapus dulu (atau nilai *a*-nya diganti).

Anda dapat memberi nama tertentu untuk constraint jika diinginkan (melalui klausa CONSTRAINT *nama_constraint*). Jika tidak diberikan maka database akan memberi nama otomatis untuk constraint ini. Catatan untuk MySQL: Jika disebutkan sebagai constraint kolom (bukan constraint tabel), maka Anda tidak bisa memberi nama sendiri untuk constraint.

Unique constraint dan NULL: Sebuah kolom yang diberi constraint UNIQUE masih dapat mengandung nilai NULL (kecuali jika juga diberi constraint NOT NULL). Yang perlu diperhatikan adalah, meskipun kolom diberi constraint UNIQUE, namun nilai NULL dapat ada beberapa! Dengan kata lain, jika sebelumnya ada baris yang memiliki nilai *a*=NULL, maka boleh-boleh saja ada baris baru yang ditambahkan dengan *a*=NULL lagi. (Catatan: ada sebagian database yang tidak berkelakuan standar dan membatasi hanya satu nilai NULL. MySQL termasuk mengikuti kelakuan standar SQL dalam hal ini, yaitu membolehkan multiple NULL dalam kolom UNIQUE).

Ini juga sejalan dengan definisi NULL itu sendiri yaitu sebuah nilai yang tidak diketahui dan tidak sama dengan semua nilai lainnya, termasuk dengan NULL itu sendiri. Jika Anda tidak menginginkan adanya NULL di dalam kolom Anda, tentu saja tinggal menambahkan constraint NOT NULL.

Unique constraint vs unique index: Dalam database dikenal juga istilah unique index. Sebetulnya kedua hal ini berbeda, namun memang semua database menggunakan unique index untuk mengimplementasi unique constraint. Jika sebuah kolom didefinisikan sebagai UNIQUE, maka database akan secara otomatis membuatkan unique index untuk kolom tersebut. Indeks digunakan untuk mempercepat pengecekan apakah sebuah nilai yang ingin dimasukkan sudah ada sebelumnya pada tabel. Namun kalau kita berbicara dari sisi SQL yang high level, yang kita peduli hanyalah istilah unique constraint, sementara unique index merupakan "implementation detail."

Sayangnya, MySQL cenderung mengaburkan perbedaan ini dengan menganggap UNIQUE KEY dan UNIQUE INDEX sebagai sinonim. Di database lain, keduanya lebih jelas bedanya. Menghapus indeks belum tentu otomatis menghapus constraintnya.

Unique constraint dapat didefinisikan sebagai constraint kolom (langsung mengikuti definisi sebuah kolom) atau sebagai constraint tabel (disebutkan terpisah dan dapat menyebutkan lebih dari satu kolom sekaligus). UNIQUE (a, b) artinya kombinasi nilai a dan b harus unik. Dengan kata lain, boleh ada baris dengan dua buah nilai a yang sama atau dua buah nilai b yang sama, tapi hanya boleh ada satu kombinasi (a, b) yang sama.

Primary key constraint

```
-- Resep 4-2-3: primary key constraint
CREATE TABLE ... (
  ...
 -- sebagai constraint kolom
  nama_kolom tipe_data [CONSTRAINT nama_constraint] PRIMARY KEY
  ...
  -- sebagai constraint tabel
  [CONSTRAINT nama_const] PRIMARY KEY (nama_kolom[, nama_kolom2,
...])
  ...
);
```

PRIMARY KEY (PK) constraint pada dasarnya sama artinya dengan UNIQUE NOT NULL (yakni, kolom harus unik dan juga tidak boleh null). Perbedaannya dengan UNIQUE NOT NULL, dalam sebuah tabel hanya boleh ada satu PK constraint sementara UNIQUE constraint boleh ada banyak.

Catatan: ada beberapa database tertentu (termasuk versi lama MySQL) yang tidak menerima sintaks "PRIMARY KEY" saja, melainkan harus "NOT NULL PRIMARY KEY". Padahal menurut standar, PRIMARY KEY otomatis berarti NOT NULL.

Menurut prinsip database relasional yang dikemukakan Codd, setiap tabel harus memiliki PK agar tiap baris dapat diidentifikasi melalui PK-nya (karena PK unik untuk setiap baris). SQL tidak mewajibkan setiap tabel yang kita buat dengan CREATE TABLE mengandung PK, hanya jika kita definisikan saja PK-nya.

Sama seperti unique constraint, PK constraint juga diimplementasi di database menggunakan unique index. Tapi perlu diingat bahwa keduanya berbeda. Constraint adalah model logis untuk database, sementara index adalah model fisik.

PK constraint dapat didefinisikan sebagai constraint kolom (langsung mengikuti definisi sebuah kolom) atau sebagai constraint tabel (disebutkan terpisah dan dapat lebih dari satu kolom sekaligus). PRIMARY KEY (a, b) artinya kombinasi nilai a dan b harus unik (plus, baik a maupun b tidak boleh mengandung NULL). Dengan kata lain, boleh ada baris dengan dua buah nilai a yang sama atau dua buah nilai b yang sama, tapi hanya boleh ada satu kombinasi (a, b) yang sama.

Catatan untuk MySQL: Jika disebutkan sebagai constraint kolom, maka Anda tidak bisa memberi nama sendiri untuk constraint.

Foreign key constraint

```
-- Resep 4-2-4: foreign key constraint
CREATE TABLE ... (
  ...

  -- sebagai constraint kolom
  nama_kolom tipe_data [CONSTRAINT nama_const]
    REFERENCES nama_tabel(kolom1[, kolom2, ...])
    [ON UPDATE  RESTRICT|CASCADE|SET NULL|NO ACTION|SET DEFAULT]
    [ON DELETE  RESTRICT|CASCADE|SET NULL|NO ACTION|SET DEFAULT]
  ...

  -- sebagai constraint tabel
  [CONSTRAINT nama_const] FOREIGN KEY
    (nama_kolom[, nama_kolom2, ...])
    REFERENCES nama_tabel(kolom1[,kolom2, ...])
    [ON UPDATE  RESTRICT|CASCADE|SET NULL|NO ACTION|SET DEFAULT]
    [ON DELETE  RESTRICT|CASCADE|SET NULL|NO ACTION|SET DEFAULT]
  ...

);
```

Foreign key (FK) constraint menjaga/membatasi agar nilai kolom harus terdapat pada kolom pada tabel lain. FK constraint merupakan sesuatu yang amat penting dalam database relasional, yaitu untuk menjaga "referential integrity". Mengulangi penjelasan di Bab 2, maksud referential integrity adalah, data yang dependen (bergantung pada data lain, merujuk/refer ke data lain) selalu dijaga agar nilainya selalu ada pada data yang dirujuk. Kalau dianalogikan dengan bahasa pemrograman C, data yang dependen ini adalah seperti pointer, dan FK constraint menjaga agar nilai pointer ini selalu merujuk pada data yang valid.

Sebuah contoh:

```
-- tabel daftar negara
CREATE TABLE negara (
  kode CHAR(3) PRIMARY KEY, – kode ISO 3 huruf
  nama VARCHAR(64) NOT NULL
);
-- isi tabel daftar negara
INSERT INTO negara VALUES ('CHN', 'China');
```

```
INSERT INTO negara VALUES ('INA', 'Indonesia');
INSERT INTO negara VALUES ('IND', 'India');

-- tabel ibukota negara
CREATE TABLE ibukota (
  kodenegara CHAR(3) PRIMARY KEY REFERENCES negara(kode),
  ibukota VARCHAR(64) NOT NULL
);
-- isi tabel ibukota negara
INSERT INTO ibukota VALUES ('CHN', 'Beijing');
INSERT INTO ibukota VALUES ('INA', 'Jakarta');
INSERT INTO ibukota VALUES ('IND', 'New Delhi');
```

Pada contoh di atas, tabel *ibukota* memiliki kolom *kodenegara* yang merupakan PRIMARY KEY sekaligus juga FOREIGN KEY. Nilai pada kolom *kodenegara* ini haruslah salah satu dari nilai kode negara yang ada pada tabel *negara* (dalam kasus ini hanyalah salah satu dari 'CHN', 'INA', atau 'IND'). Kalau kita melakukan insert sebagai berikut:

```
INSERT INTO ibukota VALUES ('JPN', 'Tokyo');
```

Maka database akan menolaknya dengan pesan kesalahan bahwa FK constraint terlanggar. Agar data bisa masuk, kita harus melakukan insert dulu negara Jepang ke tabel *negara*:

```
INSERT INTO negara VALUES ('JPN', 'Japan');
```

Barulah insert ke tabel *ibukota* akan diterima, karena 'JPN' kini merupakan salah satu nilai yang ada pada tabel *negara*. Sehingga FK constraint tidak terlanggar. Dari sini bisa jelas terlihat bahwa FK constraint menjaga agar data tetap konsisten. Kita tidak bisa sembarangan memasukkan kode negara asal ke tabel *ibukota*. Dengan kata lain, ibukota yang dimasukkan haruslah negara yang sudah dikenal di tabel *negara*.

FK constraint dapat diterapkan sebagai constraint kolom (langsung mengikuti definisi kolom yang bersangkutan) atau sebagai constraint tabel (di mana kita dapat menyebutkan kombinasi lebih dari satu kolom untuk merujuk ke kombinasi kolom di tabel lain).

Catatan untuk MySQL: FK constraint menurut saya merupakan sesuatu yang amat penting dalam menjaga integritas database, bahkan integral posisinya dalam pemodelan database relasional. Namun sayangnya, sejak bertahun-tahun lalu para

pengembang MySQL tidak memperhatikan hal ini. Dalam manual MySQL, pengembang MySQL berkata bahwa FK tidak diperlukan dan hanya memperlambat database saja. Memang betul, kita tidak *wajib* menambahkan FK constraint pada setiap kolom yang dependen. Dan memang FK constraint akan memperlambat proses insert dan update, dikarenakan untuk setiap perubahan tersebut database perlu mengecek dulu nilai kolom FK pada tabel yang dirujuk. Namun seharusnya integritas data lebih dipentingkan ketimbang kecepatan (yang dapat ditingkatkan dengan mesin/prosesor yang lebih cepat, disk yang lebih cepat, dan seterusnya).

Baru akhir-akhir ini saja MySQL mengimplementasi FK constraint, dan baru pada tabel yang engine-nya InnoDB saja. Engine tabel MyISAM belum mendukung FK constraint checking sama sekali. Ini artinya, jika Anda membuat tabel berjenis MyISAM dan mendeklarasikan FK constraint, maka FK constraint tersebut akan diabaikan dan Anda tetap bisa melanggarnya tanpa MySQL mengeluarkan pesan peringatan atau kesalahan.

Cascade update dan delete. Perlu diakui, kehadiran foreign key constraint memberikan sedikit "hambatan" atau ketidaknyamanan dalam operasi database. Kita tidak dapat menghapus tabel jika tabel yang bersangkutan masih dirujuk oleh foreign key di tabel lain (dengan kata lain, masih menjadi tabel "induk" bagi tabel lain). Demikian pula dengan menghapus baris yang mengandung kolom yang dirujuk oleh foreign key.

Untuk memperingan kerepotan ini, di SQL diperkenalkan konsep cascade delete dan update. Perhatikan klausa opsional ON UPDATE ... dan ON DELETE ... pada Resep 4-2-4. Defaultnya, jika tidak kita sebutkan, ON UPDATE/DELETE NO ACTION. Artinya, jika kita mencoba menghapus atau mengubah sebuah baris yang dirujuk foreign key, maka server database akan menolaknya dengan pesan kesalahan. Tapi jika kita menyebutkan action CASCADE, maka kita bisa menyuruh database untuk ikut menghapus para foreign key yang merujuk ke baris yang ingin kita hapus. Misalnya, kita memiliki tabel:

```
CREATE TABLE pesanan (
  id INT PRIMARY KEY,
  tanggal DATE,
  nama_pemesan VARCHAR(128),
  alamat1 VARCHAR(128),
  alamat2 VARCHAR(128),
  kota VARCHAR(64),
  propinsi VARCHAR(64),
  kodepos CHAR(5)
```

```
) ENGINE=InnoDB;

CREATE TABLE detail_pesanan (
  id_pesanan INT,
    INDEX (id_pesanan),
    FOREIGN KEY (id_pesanan) REFERENCES pesanan(id)
      ON UPDATE CASCADE
      ON DELETE CASCADE
  nama_barang VARCHAR(128),
  jumlah SMALLINT,
  harga NUMERIC(18,4)
);
```

Setiap pesanan (setiap baris di tabel *pesanan*) berisi satu atau lebih barang, yang masing-masing dicatat dalam tabel *detail_pesanan*. Di sini bisa dikatakan bahwa tabel *pesanan* adalah tabel induk untuk *detail_pesanan*. Tanpa klausa ON DELETE CASCADE pada foreign key constraint, maka jika kita mencoba menghapus sebuah pesanan (sebuah baris di tabel *pesanan*) maka akan gagal karena masih ada detail pesanan di tabel *detail_pesanan*. Namun dengan klausa cascade untuk delete, jika kita menghapus sebuah pesanan maka otomatis semua detail pesanan di tabel *detail_pesanan* untuk pesanan yang bersangkutan akan ikut terhapus. Demikian pula untuk cascade update, jika kita mengubah kolom id sebuah pesanan, maka id di kolom *detail_pesanan* akan ikut berubah.

Melihat kenyamanan ini, maka Anda mungkin berpikir semua foreign key ditambahi klausa ON UPDATE CASCADE dan ON DELETE CASCADE saja. Namun sebetulnya kelakuan cascade ini tidak selalu cocok/tidak selalu yang kita inginkan. Untuk hubungan yang bersifat induk-anak memang menghapus induk sebaiknya langsung menghapus anak. Namun tidak semua hubungan bersifat induk-anak. Jika tabel *detail_pesanan* pada contoh sebelumnya berisi id_produk yang merujuk pada PK tabel *produk*, maka menurut saya tidaklah bijaksana jika kita tambahkan ON DELETE CASCADE di sana. Justru foreign key di sini sebaiknya mencegah adanya produk yang dihapus di tabel *produk* seandainya produk tersebut ada disebutkan dalam sebuah pesanan. Demikian pula untuk kasus-kasus lainnya di mana hubungan foreign key adalah lebih ke tabel data referensi (mis: tabel daftar kota/propinsi/negara, tabel daftar mata uang, tabel daftar user, dan lain sebagainya). Sebaiknya, saran saya, foreign key di sini tidak ON DELETE CASCADE untuk mencegah data yang masih dibutuhkan (masih dirujuk) terhapus.

Selain kelakuan CASCADE, database SQL memberikan opsi action lain yaitu RESTRICT (pada dasarnya sama dengan NO ACTION, melarang baris yang dirujuk

untuk dihapus/diubah, dengan perbedaan tidak dapat didefer/dimatikan), SET NULL (mengubah kolom yang merujuk menjadi NULL, tapi ini tidak selalu dapat dilakukan jika si kolom yang merujuk didefinisikan NOT NULL), SET DEFAULT (mengubah kolom yang merujuk menjadi nilai defaultnya, ini juga tidak selalu dapat dilakukan apabila nilai defaultnya melanggar foreign key constraint).

CHECK constraint

```
-- Resep 4-2-5: CHECK constraint
-- NOTE: Hanya valid untuk 4.0 ke atas dan tidak
--   diimplementasi di MySQL saat ini
CREATE TABLE ... (
  ...
  -- sebagai constraint kolom
  nama_kolom tipe_data [CONSTRAINT nama_const] CHECK (ekspresi)
  ...
  -- sebagai constraint tabel
  [CONSTRAINT nama_const] CHECK (ekspresi)
  ...
);
```

CHECK constraint juga merupakan sesuatu yang amat penting untuk integritas data. CHECK constraint adalah pengecekan yang bersifat general, yaitu untuk menjaga agar nilai sebuah kolom atau kombinasi kolom memenuhi ekspresi/ rumus/persyaratan tertentu.

Catatan untuk MySQL: MySQL menerima sintaks CHECK constraint (kecuali sintaks nama custom untuk constraint kolom) namun sangat disayangkan MySQL belum mengimplementasinya sama sekali. Artinya, constraint yang Anda definisikan akan diabaikan begitu saja.

Beberapa contoh CHECK constraint:

```
-- Contoh CHECK constraint 1
CREATE TABLE t1 (i INT NOT NULL CHECK i >= 0);
```

Kolom *i* pada tabel *t1* hanya dapat menerima bilangan cacah (bilangan bulat yang tidak negatif). Jika dimasukkan nilai *i* negatif maka database akan menolak dengan pesan kesalahan bahwa CHECK constraint terlanggar. Di MySQL Anda dapat mensimulasi ini menggunakan tipe data INT UNSIGNED namun hal ini membuat database Anda menjadi tidak portabel.

```
-- Contoh CHECK constraint 2
CREATE TABLE "dua dadu" (
  dadu1 SMALLINT NOT NULL CHECK (dadu1 BETWEEN 1 AND 6),
  dadu2 SMALLINT NOT NULL CHECK (dadu2 BETWEEN 1 AND 6)
);
```

Tabel *dua dadu* di atas mencatat pelemparan dua buah dadu. Setiap hasil pelemparan dadu dicatat pada kolom *dadu1* dan *dadu2*. Hasil pelemparan tentu saja hanyalah salah satu dari 1, 2, 3, 4, 5, atau 6.

Di MySQL Anda dapat mensimulasi hal ini dengan menggunakan tipe data ENUM:

```
-- Contoh CHECK constraint 2, versi MySQL
CREATE TABLE 'dua dadu' (
  dadu1 ENUM('1', '2', '3', '4', '5', '6') NOT NULL,
  dadu2 ENUM('1', '2', '3', '4', '5', '6') NOT NULL
);
```

Sayangnya, ENUM bukanlah constraint sejati. Kita masih dapat memasukkan nilai nonvalid seperti '7' atau 'hehehe', namun akan diubah menjadi NULL. Jadi kolom bertipe ENUM mau tidak mau harus menerima NULL, dan constraint NOT NULL yang kita definisikan akan diabaikan.

```
-- Contoh CHECK constraint 3
CREATE TABLE t2 (
  i INT NOT NULL,
  j INT NOT NULL,
  k INT NOT NULL,
  CHECK (i+j+k = 0)
);
```

Pada contoh di atas, kita memiliki sebuah CHECK constraint tabel untuk *t2*. Nilai baris yang dapat dimasukkan antara lain (1, 0, -1), (1, 2, -3), atau (0, 0, 0). Baris-baris yang jumlah kolom *i*+*j*+*k*-nya tidak nol akan ditolak, mis: (1, 1, 1), (1, 2, 3), dan lain sebagainya. Di MySQL hal ini tidak dapat dilakukan, jadi pengecekan mau tidak mau harus dilakukan di sisi aplikasi.

```
-- Contoh CHECK constraint 4
CREATE TABLE even (
  id INT PRIMARY KEY,
```

```
  keterangan VARCHAR(128) NOT NULL,
  tanggalmulai DATE NOT NULL,
  tanggalselesai DATE NOT NULL,
  CHECK (tanggalselesai >= tanggalmulai)
);
```

Contoh di atas memperlihatkan CHECK constraint untuk tipe data tanggal. Di MySQL hal ini tidak dapat dilakukan, jadi pengecekan mau tidak mau harus dilakukan di sisi aplikasi.

```
-- Contoh CHECK constraint 5
CREATE TABLE ibukota (
  kodenegara CHAR(3) PRIMARY KEY,
    CHECK (EXISTS (SELECT kode FROM negara WHERE
kode=NEW.kodenegara)),
  ibukota VARCHAR(64) NOT NULL
);
```

Contoh di atas memperlihatkan CHECK constraint dengan ekspresi berupa query SQL. Tidak semua database mendukung sintaks ini, misalnya PostgreSQL belum mendukung query dalam ekspresi CHECK.

Dari beberapa contoh di atas, bisa dilihat bahwa CHECK constraint bersifat generik sebab ekspresi CHECK constraint dapat berupa ekspresi apa saja, termasuk query SQL. Bahkan sebetulnya CHECK constraint dapat dipakai untuk mengimplementasi jenis-jenis constraint lainnya. Misalnya, untuk mengimplementasikan constraint NOT NULL:

```
-- Menggunakan CHECK constraint untuk mengimplementasi constraint
-- NOT NULL
CREATE TABLE t (
  k INT CHECK (k IS NOT NULL)
);
```

CHECK constraint untuk mengimplementasikan unique constraint:

```
-- Menggunakan CHECK constraint untuk mengimplementasi unique
-- constraint
CREATE TABLE t (
  k INT CHECK (NOT EXISTS (SELECT k FROM t WHERE t.k = NEW.k))
);
```

CHECK constraint untuk mengimplementasikan foreign key constraint:

```
-- Menggunakan CHECK constraint untuk mengimplementasi foreign key
-- constraint
CREATE TABLE t (
  k INT CHECK (NOT EXISTS (SELECT k FROM t2 WHERE t2.k = NEW.k))
);
```

Kesimpulan mengenai constraint

MySQL memiliki keterbatasan dalam hal constraint. Dua constraint yang penting belum diimplementasi sepenuhnya: FK constraint baru ada pada engine InnoDB dan CHECK constraint belum diimplementasi sama sekali. Kekurangan ini memaksa programer aplikasi untuk melakukan pengecekan secara manual dari sisi aplikasi, misalnya pada skrip PHP. Hal ini memiliki kelemahan yaitu masih adanya kemungkinan lolos. Kalau diibaratkan, pengecekan constraint oleh database itu seperti lubang pintu yang memiliki pintu dengan security card otomatis. Sementara pengecekan dari sisi aplikasi seperti lubang pintu yang tidak memiliki pintu sehingga harus dijaga oleh orang. Kerugiannya: 1) “capek”, kita harus menunggui pintu dan menanyai setiap orang yang ingin lewat (menulis baris pengecekan setiap kali ingin menginput data ke database); 2) kadang-kadang si orangnya tertidur, “meleng”, atau “ngeloyor” entah ke mana (kadang programer aplikasi lupa, menulis kode pengecekan yang salah, atau malas; atau kadang-kadang ada orang yang memasukkan data langsung ke database tanpa melalui aplikasi, misalnya melalui klien console `mysql`). Pengecekan constraint memang sebaiknya sebisa mungkin dilakukan di database.

Dengan keberadaan constraint CHECK Anda bisa dengan amat fleksibel menentukan segala macam restriksi atau persyaratan nilai yang dapat masuk ke sebuah tabel sesuai keperluan aplikasi (“business requirements”) Anda.

Saya amat menyarankan Anda menambahkan sebanyak mungkin constraint sebagaimana perlu pada tabel-tabel Anda. Seminimalnya, untuk setiap kolom Anda perlu bertanya 3 hal ini: a) apakah nilai NULL diperbolehkan? (jika tidak, berarti tambahkan constraint NOT NULL); b) apakah nilai duplikat diperbolehkan? (jika tidak, berarti tambahkan unique constraint atau primary key constraint; c) apakah nilai ini bergantung pada tabel lain? (jika ya, tambahkan foreign key constraint). Menjaga sebuah tabel dari data yang ngaco atau tidak relevan pada umumnya lebih penting daripada berusaha membuat segalanya secepat mungkin. Peningkatan kecepatan mudah diperoleh dengan upgrade hardware, tapi data yang salah amat merepotkan bahkan kadang tak memungkinkan lagi untuk diperbaiki.

4.3 Domain

```
-- Resep 4-3-1: Deklarasi domain
-- NOTE: belum didukung MySQL
CREATE DOMAIN nama_domain AS tipe_data
  DEFAULT ekspresi
  [constraint [...]]
```

Domain pada dasarnya adalah definisi tipe data baru, yang terdiri dari tipe data basis sebagai dasar plus constraint-constraint yang mungkin ingin diterapkan pada tipe data tersebut.

Domain berguna saat kita ingin menyebutkan data-data sejenis lebih di satu tempat. Dengan domain, tak perlu menyebutkan constraint yang sama berulang-ulang di tiap definisi tabel. Domain hasil definisi kita ini nantinya dapat disebutkan sebagai tipe data pada CREATE TABLE.

Domain belum didukung di MySQL namun berikut ini beberapa beberapa contoh deklarasi domain:

```
-- Contoh 4-3-1
CREATE DOMAIN tipe_uang AS DECIMAL(18,4);

CREATE DOMAIN tipe_jenis_kelamin AS CHAR(1)
  CHECK (jenis_kelamin='L' OR jenis_kelamin='P');
```

Domain-domain ini nantinya dapat dipakai di dalam deklarasi tabel:

```
-- Contoh 4-3-2
CREATE TABLE detil_transaksi (
  account_dari INT NOT NULL REFERENCES account(id),
  account_ke INT NOT NULL REFERENCES account(id),
  jumlah TIPE_UANG NOT NULL
);

CREATE TABLE karyawan (
  nama VARCHAR(128) NOT NULL,
  kelamin TIPE_JENIS_KELAMIN NOT NULL
);
```

Constraint-constraint yang terdefinisi di deklarasi domain akan ikut dicek beserta dengan constraint yang Anda definisikan pada kolom atau tabel. Contohnya, pada tabel *karyawan*, selain *jenis_kelamin* harus satu huruf dan hanya dapat bernilai 'L' atau 'P', kolom tersebut juga tidak boleh NULL.

4.4 Beberapa Resep CREATE TABLE Lain

Ada beberapa poin penting lagi yang perlu Anda ketahui dalam membuat tabel di SQL.

Serial/Autoincrement

```
-- Resep 4-4-1: auto_increment
CREATE TABLE ... (
  ...
  nama_kolom INT [PRIMARY KEY | UNIQUE] AUTO_INCREMENT
  ...
);
-- Resep 4-4-2: SERIAL
-- NOTE: Hanya di MySQL 4.1
CREATE TABLE ... (
  ...
  nama_kolom SERIAL
  ...
);
```

Sebuah kolom numerik yang didefinisikan sebagai unique constraint atau primary key constraint dapat diberi opsi AUTO_INCREMENT. Artinya, jika dimasuki nilai NULL, maka server database akan otomatis memberikan nilai yang besarnya 1 lebih besar dari nilai terbesar yang ada pada kolom tersebut (atau nilai 1 jika ini adalah baris pertama yang dimasukkan).

SERIAL diperkenalkan di MySQL 4.1 dan merupakan sinonim untuk BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE. SERIAL juga dikenal di beberapa database lain dan kini menjadi bagian dari standar SQL:2003.

Serial atau auto_increment berguna untuk memberikan nilai urut unik tanpa kita harus mengingat urutannya. Misalnya, kita bisa memasukkan empat buah baris tanpa menyebutkan nilai kolom auto_increment, dan nanti nilainya otomatis menjadi 1, 2, 3, 4.

Tanpa serial atau auto_increment, dalam memberi nilai unik kadang kita harus menggunakan tanggal/timestamp, kombinasi nomor random, kode tertentu buatan manusia, atau GUID (nilai 128bit).

Membuat Tabel Hanya Jika Sebelumnya Tidak Ada

MySQL menyediakan sintaks yang memudahkan kita membuat tabel hanya jika tabel tersebut sebelumnya tidak ada. Sintaks ini tidak ada di standar SQL, tapi amat berguna. Di database lain untuk melakukan hal yang serupa harus dilakukan lewat bahasa pemrograman (dengan mengecek keberadaan tabel menggunakan API atau metadata) atau menggunakan stored procedure. Sementara di MySQL cukup dengan statement SQL:

```
-- Resep 4-4-3: Membuat tabel hanya jika sebelumnya tidak ada
CREATE TABLE IF NOT EXISTS (...);
```

Perintah di atas berguna biasanya dalam deretan perintah SQL yang dijalankan secara batch terhadap database yang mungkin sudah berisi. Jika tabel sudah ada, maka perintah CREATE TABLE akan diabaikan dan tidak mengeluarkan error (tetap dianggap berhasil). Jika tidak menyertakan klausa IF NOT EXISTS maka perintah akan gagal.

Membuat Tabel Seperti Tabel Lain

```
-- Resep 4-4-4: Membuat tabel dengan definisi dari tabel lain
-- NOTE: Hanya di MySQL 4.1
CREATE TABLE nama_tabel LIKE nama_tabel_lain;
```

Dengan sintaks di atas tersebut, kita dapat membuat sebuah tabel yang strukturnya (dan termasuk constraint-constraintnya) sama dengan sebuah tabel yang sudah ada. Ini berguna misalnya jika kita ingin mencoba-coba mengutak-atik sebuah tabel tapi tidak ingin melakukannya pada tabel yang sebenarnya.

CREATE TABLE LIKE dikenalkan di standar SQL:1999 dan diperluas di SQL:2003 yaitu dapat menambah kolom baru (“buat seperti tabel T plus kolom a, b, c”).

Melihat Daftar Tabel

```
-- Resep 4-4-5: Melihat daftar tabel
SHOW TABLES;
```

Contoh:

```
mysql> show tables;
+---------------+
| Tables_in_test |
+---------------+
|  dua          |
| Accidents     |
| create        |
| dua           |
| dua-juga      |
| person        |
| person_likes  |
| t1            |
| t2            |
| t3            |
+---------------+
10 rows in set (0.01 sec)
```

Perintah ini dikenal di klien console `mysql` dan tidak didefinisikan oleh standar SQL. Di database lain bisa saja caranya berbeda untuk melihat daftar tabel dalam sebuah database.

Mengetahui Struktur Tabel

```
-- Resep 4-4-6: Mengetahui struktur tabel

-- menampilkan dalam bentuk tabular
DESCRIBE nama_tabel;

-- output alternatif, menampilkan perintah CREATE TABLE
SHOW CREATE TABLE nama_tabel;
```

Contoh:

```
mysql> describe article;
| Field           | Type              | Null | Key | Default           | Extra          |
+-----------------+-------------------+------+-----+-------------------+----------------+
| id              | int(10) unsigned  |      | PRI | NULL              | auto_increment |
| creatoradminuserid | varchar(32)    |      | MUL |                   |                |
| ispublic        | tinyint(3) unsigned |    |     | 0                 |                |
| title           | varchar(255)      |      | MUL |                   |                |
| subtitle        | varchar(255)      |      |     |                   |                |
| previewcontent  | text              | YES  |     | NULL              |                |
```

```
| content          | mediumtext          | YES |     | NULL                |               |
| ctime            | datetime            |     | MUL | 0000-00-00 00:00:00 |               |
| mtime            | datetime            |     | MUL | 0000-00-00 00:00:00 |               |
| categoryid       | int(11)             |     | MUL | 0                   |               |
| keywords         | varchar(255)        |     |     |                     |               |
| has_thumbnaillogo | tinyint(3) unsigned |     |     | 0                   |               |
| has_logo         | tinyint(3) unsigned |     |     | 0                   |               |
| note             | varchar(255)        |     |     |                     |               |
| publishtime      | datetime            |     | MUL | 0000-00-00 00:00:00 |               |
+------------------+---------------------+-----+-----+---------------------+---------------+
15 rows in set (0.02 sec)
```

```
mysql> show create table article;
+---------+--------------...
| Table   | Create Table  ...
+---------+--------------...
| article |CREATE TABLE `article` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `creatoradminuserid` varchar(32) NOT NULL default '',
  `ispublic` tinyint(3) unsigned NOT NULL default '0',
  `title` varchar(255) NOT NULL default '',
  `subtitle` varchar(255) NOT NULL default '',
  `previewcontent` text,
  `content` mediumtext,
  `ctime` datetime NOT NULL default '0000-00-00 00:00:00',
  `mtime` datetime NOT NULL default '0000-00-00 00:00:00',
  `categoryid` int(11) NOT NULL default '0',
  `keywords` varchar(255) NOT NULL default '',
  `has_thumbnaillogo` tinyint(3) unsigned NOT NULL default '0',
  `has_logo` tinyint(3) unsigned NOT NULL default '0',
  `note` varchar(255) NOT NULL default '',
  `publishtime` datetime NOT NULL default '0000-00-00 00:00:00',
  PRIMARY KEY  (`id`),
  KEY `creatoradminuserid` (`creatoradminuserid`),
  KEY `ctime` (`ctime`),
  KEY `mtime` (`mtime`),
  KEY `categoryid` (`categoryid`),
  FULLTEXT KEY `title`
```

```
('title','subtitle','previewcontent','content','keywords'),
  KEY 'idx1' ('publishtime')
) TYPE=MyISAM |
+------------+----------...
1 row in set (0.03 sec)
```

Perintah ini dikenal di klien console `mysql` dan tidak didefinisikan oleh standar SQL. Di database lain bisa saja caranya berbeda untuk melihat daftar tabel dalam sebuah database.

Jenis Engine

Ini spesifik di MySQL. Anda dapat memilih jenis engine yang ingin dipakai untuk sebuah tabel.

```
-- Resep 4-4-7: Memilih jenis engine untuk tabel
-- NOTE: untuk MySQL 4.0 ke bawah, ganti kata kunci ENGINE dengan
TYPE
CREATE TABLE (...) ENGINE=jenis_engine;
```

di mana jenis_engine adalah BDB (atau BerkeleyDB), MEMORY (HEAP), ISAM, InnoDB, MERGE (atau MRG_MyISAM), atau MyISAM. Defaultnya, jika tidak disebutkan, maka dianggap MyISAM. Pada MySQL versi lama (4.0 ke bawah), ganti kata kunci ENGINE dengan TYPE.

Jenis engine yang berbeda memiliki karakteristik, restriksi, atau fitur yang berbeda. Contohnya, foreign key constraint, transaksi, dan row-level locking baru didukung di engine InnoDB. InnoDB juga memiliki arsitektur multiversioning yang menguntungkan untuk kondisi klien simultan yang banyak. Sementara MyISAM merupakan engine yang sudah menjadi default sejak 3.23. Karena kematangannya, ada fitur-fitur yang hanya ada pada jenis engine ini misalnya full-text indexing dan kompresi indeks. Sayangnya saat ini MyISAM belum mendukung transaksi (COMMIT, ROLLBACK). Jenis engine tabel lain lebih jarang digunakan dan pada kondisi tertentu saja. Jenis tabel MERGE misalnya untuk membuat beberapa buah tabel MyISAM (yang strukturnya sama) dapat diakses seperti sebuah tabel tunggal saja. Tabel berjenis MEMORY sepenuhnya disimpan di memori dan bersifat tidak persisten, berguna untuk membuat tabel sementara yang dapat diakses beberapa klien sekaligus. Tabel ISAM adalah engine lama yang telah digantikan oleh MyISAM. Engine BDB adalah engine yang menggunakan software BerkeleyDB untuk mendukung transaksi, namun statusnya saat ini praktis telah tergantikan oleh InnoDB.

Pembahasan yang lebih menyeluruh mengenai karakteristik dari tiap-tiap engine dapat Anda baca pada manual MySQL.

Indeks

```
-- Resep 4-4-8: Menambahkan indeks pada tabel yang sudah ada
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX nama_indeks
  ON nama_tabel (nama_kolom[, ...]);

-- Langsung sekalian membuat indeks saat CREATE TABLE
CREATE TABLE nama_tabel (
  ...
  INDEX(nama_kolom[, ...])
  UNIQUE(nama_kolom[, ...])
  ...
);

-- Resep 4-4-9: Melihat indeks untuk sebuah table
SHOW INDEX FROM nama_tabel;
```

Indeks dapat ditambahkan pada sebuah kolom atau kombinasi kolom. Indeks dapat dibuat berbarengan saat tabel diciptakan, atau menyusul melalui CREATE INDEX. Terdapat dua golongan indeks: indeks non-unique (dibuat dengan kata kunci CREATE INDEX atau INDEX pada CREATE TABLE) dan unique index (dibuat dengan kata kunci CREATE UNIQUE INDEX atau UNIQUE pada CREATE TABLE). Unique index sama seperti indeks biasa, tapi sekaligus melarang adanya nilai duplikat pada kolom atau kombinasi kolom. Catatan: indeks FULLTEXT dan SPATIAL akan dibahas di Bab 8.

Indeks di database relasional memiliki dua fungsi utama: 1) untuk mempercepat pencarian; 2) untuk mengimplementasi unique constraint dan primary key constraint (menggunakan unique index). Namun indeks juga memiliki konsekuensi: 1) memperlambat penulisan (INSERT) data baru dan memperlambat pengubahan data (UPDATE & DELETE); 2) memakan ruang disk.

Mempercepat pencarian di sini terutama jika tabel Anda berisi banyak baris dan kolom yang ingin diindeks memiliki rentang nilai yang lebar (selectivity yang besar). Contohnya, jika Anda memiliki tabel *pelanggan* berisi puluhan ribu baris dan ingin mencari secara cepat nama seorang pelanggan. Ini karena nama pelanggan memang bervariasi (rentang nilainya besar). Dengan kata lain, indeks akan mempercepat pencarian sebuah nilai dari banyak nilai yang ada. Sementara

jika Anda mengindeks kolom *jenis_kelamin* maka indeks di sini tidak akan membantu dikarenakan hanya ada 2 nilai yang mungkin untuk jenis kelamin.

Jika Anda menambahkan unique constraint atau primary key constraint pada sebuah kolom atau kombinasi kolom, maka secara otomatis database

Saran saya: sebagai pemula Anda tidak perlu mengkuatirkan dulu masalah indeks. Cukup buatlah tabel dan pastikan Anda telah memilih jenis tipe data yang tepat dan menambahkan constraint-constraint yang diperlukan. Baru setelah tabel banyak terisi baris atau situs sudah online dan dikunjungi banyak pengunjung, Anda bisa mempelajari lebih dalam mengenai indeks lalu menambahkan indeks sebagaimana diperlukan. Penambahan indeks merupakan sesuatu yang berhubungan dengan kinerja, dan pada banyak kasus, hanya perlu dikuatirkan manakala memang muncul masalah dengan kinerja. Kecuali Anda membuat sesuatu yang berskala besar—di mana pada saat itu Anda memang sudah harus memahami soal indeks dan isu kinerja lainnya—maka yang lebih perlu Anda kuatirkan adalah kebenaran (correctness), bukan kecepatan.

Indeks perlu ditambahkan jika: 1) ukuran tabel besar (ribuan baris ke atas); 2) kolom atau kombinasi kolom yang ingin diindeks memiliki rentang nilai yang besar; 3) kolom memang sering dicari. Indeks sebaiknya dihindari atau tidak diperlukan jika: 1) ukuran tabel kecil; 2) tabel terlalu sering ditulisi, sehingga dibutuhkan kecepatan maksimum untuk penulisan dan penambahan indeks terlalu memperlambat; 3) kolom memiliki rentang nilai yang kecil (contohnya: jenis kelamin, kolom benar-salah, dan lain sebagainya). Jadi secara membabi buta menambahkan indeks pada semua kolom bukanlah tindakan yang benar. Tapi tidak memperhatikan indeks sama sekali juga akan berakibat tabel-tabel Anda yang berukuran besar amat lambat saat dipakai.

4.5 Contoh Latihan CREATE TABLE

Latihan: Daftar Nilai Siswa Sekolah Dasar

Buatlah tabel-tabel sebagai berikut: pertama, daftar siswa yang mencakup data nama, jenis kelamin, tempat dan tanggal lahir, serta tanggal mendaftar ke sekolah. Tentu saja, saat mendaftar sekolah, semua siswa perlu memberikan informasi data dirinya dengan lengkap. Kedua, daftar ruang kelas. Setiap tingkatan kelas memiliki tiga hingga empat ruangan. Ruang kelas dinamai dengan 1A, 1B, 1C, 1D, 2A, dan seterusnya. Ketiga, tabel yang mencatat siswa mana saja yang termasuk ke dalam ruang kelas. Setiap siswa terdaftar di satu tingkat dan satu ruang kelas. Keempat, daftar guru. Data yang perlu dicatat mencakup nama, jenis kelamin, tempat dan

tanggal lahir, serta tanggal mendaftar ke sekolah. Kelima, daftar mata pelajaran. Kode mata pelajaran terdiri dari 3 huruf. Keenam, kurikulum untuk sebuah tingkatan kelas (tiap tahun mungkin saja berbeda). Ketujuh, tabel yang mencatat guru mana yang mengajar sebuah mata pelajaran pada tahun ajaran tertentu. Sebuah mata pelajaran untuk sebuah tingkatan kelas diajar oleh satu guru. Kedelapan, daftar wali kelas. Seorang guru hanya boleh menjadi wali kelas untuk satu kelas saja selama 1 tahun pelajaran yang sama. Kesembilan, rapor atau daftar nilai tiap siswa (diasumsikan untuk sepanjang tahun, kita hanya membuat satu rapor saja). Nilai tiap siswa antara 0 hingga 10. Bobot setiap pelajaran sama, sehingga untuk menghitung nilai akhir rapor nanti kita tinggal melakukan perhitungan rata-rata dari setiap nilai mata pelajaran.

Catatan: database harus dapat mencatat untuk lebih dari satu tahun pelajaran dan dapat mencatat sejarah daftar murid, siswa, dan pelajaran untuk tahun-tahun lalu.

Jawaban Latihan

Kita akan membahas tabel satu demi satu.

```
-- Contoh 4-5-1
-- NOTE: klausa CHECK hanya dikenal di MySQL 4.0 ke atas
CREATE TABLE siswa (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nama VARCHAR(128) NOT NULL,
  kelamin CHAR(1) NOT NULL CHECK (kelamin='L' OR kelamin='P'),
  tempatlahir VARCHAR(64) NOT NULL,
  tgllahir DATE NOT NULL,
  tgldaftar DATE NOT NULL
) ENGINE=InnoDB;
```

Untuk jenis kelamin sebetulnya bisa kita jadikan domain, karena akan dipakai juga di tabel *guru*, namun karena domain tidak didukung MySQL kita pakai saja definisi tipe tanpa domain. Semua data kita beri constraint NOT NULL karena menurut business requirement semua data diri ini harus diisi lengkap saat seorang siswa mendaftar ke sekolah. Jika mau, Anda bisa juga misalnya menambahkan constraint tabel CHECK untuk mencegah *tgldaftar* lebih awal dari *tgllahir*. Atau untuk mencegah *tgldaftar* lebih awal dari tanggal berdirinya sekolah yang bersangkutan.

Tentu saja, constraint CHECK yang kita definisikan saat ini tidak akan diacuhkan oleh MySQL. Tapi membuat definisi CHECK tetap berguna sebagai dokumentasi atau saat kita berpindah database suatu saat nanti.

```
-- Lanjutan Contoh 4-5-1
-- NOTE: klausa CHECK hanya dikenal di MySQL 4.0 ke atas
CREATE TABLE ruangkelas (
  nama CHAR(2) PRIMARY KEY CHECK (nama REGEXP '^[1-6][A-D]$'),
  tingkat SMALLINT NOT NULL CHECK (tingkat >= 1 AND tingkat <= 6)
) ENGINE=InnoDB;
```

Di sini kita menjadikan kolom *nama* sebagai primary key, karena memang nama ruang kelas diharuskan unik. Bisa dilihat juga adanya pengecekan regex pada kolom *nama* untuk menjaga agar nama ruang kelas benar-benar valid. Pengecekan regex bersifat case-insensitive, karena secara defaultnya memang kolom di MySQL bersifat case-insensitive. Regex di MySQL dijelaskan lebih jauh di Bab 6 mengenai query. Sebetulnya juga nama ruang kelas ini akan disebutkan di tabel lain, sehingga bisa kita ekstrak definisinya menjadi domain. Tapi MySQL belum mengenal domain.

Kolom *tingkat* juga kita batasi agar hanya dapat mengandung nilai dari 1 sampai dengan 6 (tidak ada kelas minus tujuh atau sembilan belas bukan?)

```
-- Lanjutan Contoh 4-5-1
-- NOTE: klausa CHECK hanya dikenal di MySQL 4.0 ke atas
CREATE TABLE ruangkelas_siswa (
  tahun SMALLINT NOT NULL
    CHECK (tahun >= 2000 AND tahun <= 2100),
  id_siswa INT NOT NULL,
    INDEX (id_siswa),
    FOREIGN KEY (id_siswa) REFERENCES siswa(id),
  ruangkelas CHAR(2) NOT NULL,
    INDEX (ruangkelas),
    FOREIGN KEY (ruangkelas) REFERENCES ruangkelas(nama),
  UNIQUE (tahun, id_siswa, ruangkelas)
) ENGINE=InnoDB;
```

Di sini jelaslah mengapa kita mendefinisikan tabel-tabel kita dengan engine InnoDB, yaitu karena kita ingin membuat foreign key, yang baru didukung oleh engine tersebut. Kolom *id_siswa* pada tabel *ruangkelas_siswa* merujuk pada kolom *id* (PK) tabel *siswa*. Demikian pula *ruangkelas* pada tabel *ruangkelas_siswa* merujuk pada kolom *nama* (PK) tabel *ruangkelas*. MySQL mewajibkan kedua tabel yang ingin berhubungan foreign key harus sama-sama bertipe InnoDB, lalu kedua kolom yang berhubungan juga harus diberi indeks. Terakhir, kita memberi unique constraint pada kombinasi kolom (*tahun*, *id_siswa*, *ruangkelas*). Ini untuk

memenuhi persyaratan bahwa seorang siswa hanya boleh terdaftar pada satu ruang kelas saja dalam satu tahun ajaran.

Catatan: MySQL menyediakan tipe data YEAR, tapi demi portabilitas saya menghindarinya dan sebagai gantinya menggunakan SMALLINT saja.

```
-- Lanjutan Contoh 4-5-1
-- NOTE: klausa CHECK hanya dikenal di MySQL 4.0 ke atas
CREATE TABLE guru (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nama VARCHAR(128) NOT NULL,
  kelamin CHAR(1) NOT NULL CHECK (kelamin='L' OR kelamin='P'),
  tempatlahir VARCHAR(64) NOT NULL,
  tgllahir DATE NOT NULL,
  tgldaftar DATE NOT NULL
) ENGINE=InnoDB;
```

Sebetulnya di MySQL 4.1 kita bisa juga memberi perintah: `CREATE TABLE guru LIKE siswa;` karena kebetulan struktur tabelnya sama persis.

```
-- Lanjutan Contoh 4-5-1
CREATE TABLE pelajaran (
  kode CHAR(3) PRIMARY KEY,
  nama VARCHAR(128) NOT NULL
) ENGINE=InnoDB;
```

Sudah cukup jelas.

```
-- Lanjutan Contoh 4-5-1
-- NOTE: klausa CHECK hanya dikenal di MySQL 4.0 ke atas
CREATE TABLE kurikulum (
  tahun SMALLINT NOT NULL
    CHECK (tahun >= 2000 AND tahun <= 2100),
  kodepelajaran CHAR(3) NOT NULL,
    INDEX (kodepelajaran),
    FOREIGN KEY (kodepelajaran) REFERENCES pelajaran(kode),
  tingkat SMALLINT NOT NULL
    CHECK (tingkat >= 1 AND tingkat <= 6),
  UNIQUE (tahun, kodepelajaran, tingkat)
) ENGINE=InnoDB;
```

4.5 Contoh Latihan CREATE TABLE

Tabel kurikulum mencatat mata pelajaran apa saja yang diajarkan untuk sebuah tingkatan kelas (pada tahun tertentu). Tentu saja, unique constraint tabel menjaga agar tidak ada baris dobel (lebih dari satu mata pelajaran yang sama untuk satu tingkatan, misalnya Matematika dan Matematika lagi).

```
-- Lanjutan Contoh 4-5-1
-- NOTE: klausa CHECK hanya dikenal di MySQL 4.0 ke atas
CREATE TABLE guru_pelajaran (
  tahun SMALLINT NOT NULL
    CHECK (tahun >= 2000 AND tahun <= 2100),
  id_guru INT NOT NULL,
    INDEX (id_guru),
    FOREIGN KEY (id_guru) REFERENCES guru(id),
  kodepelajaran CHAR(3) NOT NULL,
    INDEX (kodepelajaran),
    FOREIGN KEY (kodepelajaran) REFERENCES pelajaran(kode),
  tingkat SMALLINT NOT NULL
    CHECK (tingkat >= 1 AND tingkat <= 6),
  UNIQUE (tahun, kodepelajaran, tingkat)
) ENGINE=InnoDB;
```

Tabel di atas mencatat guru mana saja yang mengajar mata pelajaran pada tahun tertentu. Guru dan kode pelajaran merujuk pada tabel *guru* dan *pelajaran*. Kita tambahkan unique constraint di akhir definisi tabel untuk menjaga tidak ada lebih dari satu guru yang mengajar satu mata pelajaran yang sama untuk satu tingkat pada tahun yang sama. Tentu saja, seorang guru dapat mengajar beberapa mata pelajaran yang berbeda atau mata pelajaran yang sama untuk tingkatan kelas yang berbeda.

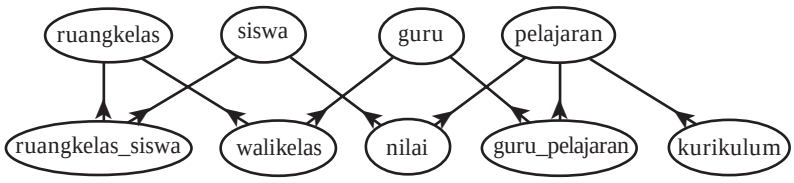
```
-- Lanjutan Contoh 4-5-1
-- NOTE: klausa CHECK hanya dikenal di MySQL 4.0 ke atas
CREATE TABLE walikelas (
  tahun SMALLINT NOT NULL
    CHECK (tahun >= 2000 AND tahun <= 2100),
  id_guru INT NOT NULL,
    INDEX (id_guru),
    FOREIGN KEY (id_guru) REFERENCES guru(id),
  ruangkelas CHAR(2) NOT NULL
    CHECK (tingkat >= 1 AND tingkat <= 6),
  UNIQUE (tahun, ruangkelas)
) ENGINE=InnoDB;
```

Sudah cukup jelas.

```
-- Lanjutan Contoh 4-5-1
-- NOTE: klausa CHECK hanya dikenal di MySQL 4.0 ke atas
CREATE TABLE nilai (
  tahun SMALLINT NOT NULL
    CHECK (tahun >= 2000 AND tahun <= 2100),
  id_siswa INT NOT NULL,
    INDEX (id_siswa),
    FOREIGN KEY (id_siswa) REFERENCES siswa(id),
  kodepelajaran CHAR(3) NOT NULL,
    INDEX (kodepelajaran),
    FOREIGN KEY (kodepelajaran) REFERENCES pelajaran(kode),
  nilai SMALLINT
    CHECK (nilai >= 0 AND nilai <= 10),
  UNIQUE (tahun, id_siswa, kodepelajaran)
) ENGINE=InnoDB;
```

Tabel terakhir di atas mencatat nilai tiap siswa untuk tiap mata pelajaran pada tahun tertentu. Unique constraint tabel menjaga agar seorang siswa tidak memiliki dua nilai untuk kode pelajaran yang sama. Sengaja kita buat kolom *nilai* dapat mengandung NULL; bayangkan situasi di mana sudah saatnya pembagian rapor, tapi ada sebuah mata pelajaran yang masih belum diketahui nilainya (mis: karena si guru sakit, atau masih ada masalah-masalah tertentu yang perlu diselesaikan seorang siswa, dan lain sebagainya).
Skema database kita kalau digambarkan hasilnya akan berbentuk seperti pada Gambar 4-5-1. Empat tabel yang berada pada deretan atas (*ruangkelas*, *siswa*, *guru*, dan *pelajaran*) bisa dibilang merupakan data *master*, karena tidak bergantung pada tabel lain melainkan sebaliknya data-data mereka dipakai oleh tabel-tabel lain.

Desain database amat bergantung pada kebutuhan (business requirement). Jika kebutuhan berbeda, bisa saja hasil akhir desain database Anda berbeda drastis dengan



Gambar 4-5-1.

yang ada pada contoh ini. Misalnya, dalam kehidupan nyata, umumnya nilai rapor dibagikan setiap 4 bulan (caturwulan) sekali atau kadang-kadang setiap 6 bulan (semester), untuk memungkinkan evaluasi sebelum akhir tahun. Atau bisa saja kode pelajaran formatnya berbeda. Atau ruang kelas dinamai secara berbeda. Atau lebih dari satu guru (guru substitusi? guru teori + guru praktik?) bisa mengajar satu mata pelajaran yang sama untuk sebuah kelas. Atau nilai sebuah mata pelajaran berupa huruf A sampai dengan F. Dan seterusnya.

4.6 ALTER TABLE

Dalam mendesain, jarang sekali biasanya sekali jalan langsung sempurna. Program memiliki nomor-nomor versi. Buku, hardware, mobil datang dalam edisi atau revisi secara regular. Demikian pula dalam merancang database. Perubahan requirement atau perbaikan desain kadang menuntut kita untuk: a) menambah atau mengurangi kolom; b) mengubah definisi kolom; c) mengganti nama kolom atau nama tabel; d) menambah atau mengurangi constraint; e) menambah atau mengurangi indeks. Bahkan kadang-kadang perubahan juga dituntut oleh masalah kinerja, masalah ruang disk, bahkan urusan nonteknis seperti marketing.

Perubahan-perubahan ini didukung oleh standar SQL melalui perintah ALTER TABLE, namun tidak semua produk database mendukung semua jenis operasi ALTER TABLE yang telah disebutkan di atas. Kadang-kadang satu-satunya cara untuk memodifikasi tabel adalah dengan membuat tabel baru, memindahkan isi dari tabel lama, dan terakhir menghapus tabel lama. Khusus di MySQL, kita kebetulan beruntung karena MySQL amat fleksibel dan mendukung semua jenis operasi perubahan tabel.

Sintaks ALTER TABLE

```
-- Resep 4-6-1: Sintaks dasar untuk mengubah struktur tabel
ALTER TABLE nama_tabel
    detil_perubahan [, detil_perubahan] ...;
```

Detil perubahan dijelaskan pada subsubbab berikut ini.

Menambah Kolom Baru

```
-- Resep 4-6-2: ALTER TABLE: menambah kolom baru
ALTER TABLE ...
  ADD [COLUMN] definisi_kolom [FIRST | AFTER nama_kolom_lain]
  ...;
```

definisi_kolom adalah seperti pada CREATE TABLE. Khusus di MySQL Anda dapat menyisipkan kolom baru ini di awal, di tengah, atau di akhir (setelah kolom-kolom tertentu yang sudah ada). Aspek pemosisian ini belum diatur oleh standar SQL dan banyak database yang tidak mendukungnya.

Mengambil contoh 4-5-1, misalnya kita ingin menambahkan kolom *orangtua* pada tabel *siswa* untuk mencatat nama orang tua murid:

```
— Contoh 4-6-1
ALTER TABLE siswa
  ADD COLUMN orangtua VARCHAR(32) NOT NULL;
```

Setelah melakukan perintah ini, maka seluruh baris pada tabel *siswa* akan memiliki nilai kolom *nama_ortu* berupa string kosong (''). Ini karena si kolom baru didefinisikan NOT NULL sehingga saat ditambahkan, MySQL mengambil nilai default untuk VARCHAR yaitu string kosong.

Menghapus Kolom

```
-- Resep 4-6-3: ALTER TABLE: menghapus kolom
ALTER [IGNORE] TABLE ...
  DROP [COLUMN] nama_kolom
  ...;
```

Jika kita menghapus sebuah kolom, maka constraint dan indeks untuk kolom bersangkutan akan turut dihapus. Menghapus kolom dapat saja gagal, misalnya kita menghapus sebuah kolom yang termasuk dalam kombinasi kolom yang didefinisikan unik atau primary key, maka setelah kolom tersebut dihapus akan terbentuk duplikat. Contohnya:

```
CREATE TABLE t1 (
  i INT NOT NULL,
  j INT NOT NULL,
  UNIQUE (i, j)
);

INSERT INTO t1 VALUES (1,1);
INSERT INTO t1 VALUES (1,2);
INSERT INTO t1 VALUES (1,3);
INSERT INTO t1 VALUES (2,1);
INSERT INTO t1 VALUES (2,2);
```

Jika kita melakukan:

```
ALTER TABLE t1 DROP j;
```

Maka akan timbul duplikat pada kolom i ((1), (1), (1), (2), (2)). Untuk mengatasinya, kita perlu mengubah dulu data tabel agar tidak timbul duplikat, atau menggunakan opsi ALTER IGNORE TABLE. Dengan opsi IGNORE, maka semua duplikat yang timbul akan dihapus otomatis. Jika kita melakukan:

```
ALTER TABLE t1 DROP j;
```

Maka hasilnya adalah:

```
mysql> select * from t1;
+---+
| i |
+---+
| 1 |
| 2 |
+---+
2 rows in set (0.02 sec)
```

dan constraintnya menjadi `UNIQUE (i)` saja.

Memodifikasi Kolom Yang Sudah Ada

```
-- Resep 4-6-4: ALTER TABLE: memodifikasi kolom yang sudah ada
ALTER [IGNORE] TABLE ...
  CHANGE [COLUMN] nama_kolom definisi_kolom
    [FIRST|AFTER nama_kolom_lain]
  ...;

-- sama dengan CHANGE, tapi MODIFY tidak bisa mengganti nama kolom
ALTER [IGNORE] TABLE ...
  CHANGE [COLUMN] definisi_kolom
    [FIRST|AFTER nama_kolom_lain]
  ...;
```

Dengan CHANGE COLUMN dan/atau MODIFY COLUMN, kita dapat mengganti nama kolom, definisinya, serta posisinya. Catatan: pengubahan posisi tidaklah diatur dalam standar SQL, jadi Anda sebaiknya tidak terlalu mementingkan posisi kolom

dalam sebuah tabel karena posisi spesifik belum tentulah didukung oleh database lain.

Contoh, kita ingin mengubah kolom *orangtua* menjadi *nama_ortu* pada tabel *siswa* dan sekalian menambah panjang maksimum stringnya:

```
ALTER TABLE siswa
  CHANGE COLUMN orangtua nama_ortu VARCHAR(128) NOT NULL;
```

Dalam mengubah kolom pun ada beberapa isu yang harus Anda perhatikan, misalnya apakah akan memotong nilai yang sudah ada (misalnya, mengubah dari VARCHAR(100) menjadi VARCHAR(50); atau INT menjadi SMALLINT), apakah akan menimbulkan duplikat (misalnya, kolom string Anda pendekkan sehingga dua buah kolom string yang panjang menjadi terpotong dan menjadi sama nilainya), dan lain sebagainya. Kalau belum yakin, ada baiknya dilakukan dulu pada tabel kosong atau database di komputer lokal untuk coba-coba.

Menambah Constraint

```
-- Resep 4-6-5: ALTER TABLE: menambah constraint

-- menambah primary key constraint
ALTER [IGNORE] TABLE ...
  ADD [CONSTRAINT [simbol]] PRIMARY KEY (nama_kolom[,...])
  ...;

-- menambah unique constraint
ALTER [IGNORE] TABLE ...
  ADD [CONSTRAINT [simbol]] UNIQUE [nama_constraint]
(nama_kolom[,...])
  ...;

-- menambah foreign key constraint
ALTER [IGNORE] TABLE ...
  ADD [CONSTRAINT [simbol]] UNIQUE [nama_constraint]
(nama_kolom[,...])
  ...;

-- menambah CHECK constraint
-- NOTE: belum didukung oleh MySQL, CHECK constraint memang belum
--         diimplementasi oleh MySQL
ALTER [IGNORE] TABLE ...
```

```
  ADD [CONSTRAINT [simbol]]
      FOREIGN KEY [nama_constraint] (nama_kolom[,...])
      REFERENCES nama_tabel(nama_kolom[,...])
  ...;
```

Menghapus Constraint

```
-- Resep 4-6-6: ALTER TABLE: menghapus constraint

-- menghapus primary key
ALTER [IGNORE] TABLE ...
  DROP PRIMARY KEY
  ...;

-- menghapus foreign key constraint
ALTER [IGNORE] TABLE ...
  DROP FOREIGN KEY nama_constraint
  ...;

-- menghapus constraint secara umum (FK, PK, CHECK, dan lain
sebagainya) berdasarkan
--   nama constraint
-- NOTE: tidak didukung MySQL. Ini bisa disebabkan karena MySQL
baru
--   mendukung unique constraint, PK, dan FK. Sementara NOT NULL
--   constraint tidak bisa diberi nama.
ALTER [IGNORE] TABLE ...
  DROP CONSTRAINT nama_constraint
  ...;
```

Catatan: MySQL belum mendukung sintaks yang generik untuk menghapus constraint (ALTER TABLE … DROP CONSTRAINT). Ini karena MySQL baru mendukung NOT NULL constraint, PK, FK, dan unique constraint. NOT NULL constraint tidak dapat diberi nama di MySQL, jadi untuk mengubah sebuah kolom menjadi nullable atau not nullable, Anda menggunakan ALTER TABLE ... CHANGE COLUMN. Untuk PK dan FK ada sintaks tersendiri (ALTER TABLE ... DROP PRIMARY KEY dan ALTER TABLE ... DROP FOREIGN KEY). CHECK constraint belum didukung. Dan untuk unique constraint, karena MySQL mengaburkan konsep unique index dan unique constraint, maka untuk menghapus unique constraint kita cukup menghapus unique indexnya. Maka sekaligus unique constraintnya juga akan terangkat. Lihat subbab-subbab selanjutnya untuk cara

menghapus dan menambah indeks.

Menambah dan Menghapus Indeks

```
-- Resep 4-6-7: ALTER TABLE: menambah dan menghapus indeks

-- menambah indeks (non-unique)
ALTER TABLE ...
  ADD INDEX [nama_indeks] (nama_kolom[, ...])
  ...;

-- menambah indeks unique (= menambah unique constraint di MySQL)
ALTER [IGNORE] TABLE ...
  ADD UNIQUE [nama_constraint] (nama_kolom[, ...])
  ...;

-- menghapus indeks atau unique constraint
ALTER TABLE ...
  DROP INDEX [nama_indeks_atau_unique_constraint]
  ...;
```

Catatan: MySQL mengizinkan kita membuat lebih dari satu indeks untuk kolom yang sama. Dua atau lebih indeks untuk kolom atau kombinasi kolom yang sama persis merupakan penyia-nyiaan ruang disk saja. Jadi Anda sebaiknya mengecek dulu apakah memang sudah ada indeks pada kolom yang ingin Anda tambahi indeks (dengan perintah SHOW INDEX, Resep 4-4-9). Ingat juga bahwa sebuah kolom yang didefinisikan PK dan UK otomatis sudah diberi indeks oleh MySQL.

Mengubah Nilai Default Kolom

```
-- Resep 4-6-8: ALTER TABLE: mengubah nilai default kolom

-- mengganti nilai default
ALTER [IGNORE] TABLE ...
  ALTER COLUMN nama_kolom
  SET DEFAULT nilai_literal
  ...;

-- menghapus nilai default
ALTER [IGNORE] TABLE ...
  ALTER COLUMN nama_kolom
```

```
  DROP DEFAULT
  ...;
```

Catatan: sebagai alternatif, untuk mengubah nilai default (dan aspek lain seperti tipe data, constraint, dan lain sebagainya), Anda juga bisa menggunakan perintah ALTER TABLE ... CHANGE COLUMN yang lebih generik.

Mengganti Nama Tabel

```
-- Resep 4-6-9: ALTER TABLE: mengganti nama tabel

-- cara pertama, sesuai standar SQL
ALTER TABLE ...
  RENAME [TO] nama_baru_tabel
  ...;

-- cara kedua, spesifik di MySQL
RENAME TABLE nama_lama TO nama_baru[, ...];
```

Cara yang sesuai standar SQL adalah menggunakan ALTER TABLE ... RENAME [TO]. Tapi MySQL juga menyediakan perintah RENAME TABLE, yang sekalian bisa mengganti nama beberapa tabel sekali jalan. Jika Anda peduli pada portabilitas, gunakan saja ALTER TABLE ... RENAME [TO].

4.7 DROP TABLE

```
-- Resep 4-7-1: menghapus tabel
DROP TABLE [IF EXISTS] nama_tabel[, ...];
```

Menghapus tabel akan otomatis menghapus semua kolom, baris, constraint, dan indeks yang terasosiasi dengan tabel yang bersangkutan. Namun jika Anda menggunakan foreign key constraint dan tabel yang ingin dihapus masih dipakai oleh tabel lain, maka tabel belum dapat dihapus. Untuk menghapusnya, hapus dulu semua tabel yang memakai tabel tersebut (atau buang dulu foreign key constraintnya). Misalnya, pada contoh 4-5-1, tabel *siswa* belum dapat dihapus sebelum tabel *ruangsiswa_kelas* dan *nilai* dihapus terlebih dahulu.

Klausa IF EXISTS adalah ekstensi yang disediakan MySQL agar mengabaikan perintah DROP TABLE (tidak menghasilkan error) jika tabel tidak ada.

4.8 Contoh Latihan ALTER TABLE

Latihan: Mengubah Database Nilai Siswa

Kita akan menggunakan Contoh 4-5-1 sebagai basis dan melakukan beberapa perubahan. Perubahan yang Anda perlu lakukan: Pertama, buatlah agar nilai rapor berlaku untuk setiap caturwulan, bukan setiap tahun. Setiap tahun tentunya terdiri dari 3 caturwulan. Kedua, tambahkanlah atribut *aktif* dan *tglkeluar* pada tabel *guru* dan *siswa* untuk mencatat apakah seorang guru atau siswa masih aktif (masih terdaftar dalam sekolah). Seorang guru atau siswa yang sudah keluar tidak akan langsung dihapus datanya di database, melainkan hanya diflag dengan nilai kolom *aktif*=0. Ketiga, tambahkan kolom *nama_wali* dan *hubungan_wali* pada tabel *siswa*. Kolom *nama_wali* dapat mencatat nama ibu, ayah, paman, atau orang lain yang bertanggung jawab terhadap si murid. Kolom *hubungan_wali* mencatat hubungan si wali dengan si murid. Keduanya dibuat sebagai kolom teks (VARCHAR). Keempat, agar lebih sejalan dengan prinsip normalisasi, maka pisahkanlah tahun ajaran dan tingkatan kelas ke dalam tabel tersendiri, dan tabel-tabel yang merujuk pada tahun ajaran atau tingkatan kelas kita buat agar bergantung melalui foreign key.

Jawaban Latihan

Untuk perubahan pertama:

```
ALTER TABLE nilai
  ADD caturwulan SMALLINT NOT NULL
    -- CHECK (caturwulan >= 1 AND caturwulan <= 3)
    AFTER tahun
;
```

Klausa CHECK saya komen karena MySQL saat ini tidak menerimanya sebagai sintaks yang valid, padahal jika disebutkan di CREATE TABLE diterima.

```
ALTER TABLE nilai
  DROP INDEX tahun,
  ADD UNIQUE (tahun, caturwulan, id_siswa, kodepelajaran);
```

Karena dalam satu tahun bisa ada lebih dari satu caturwulan, dan tiap caturwulan dapat memiliki nilai rapor yang berbeda, maka kita perlu memodifikasi unique constraint dengan menambahkan kolom *caturwulan*. Perhatikan bahwa nama constraint *tahun* yang kita hapus itu adalah nama otomatis pemberian MySQL karena sewaktu membuatnya kita tidak menyebutkan nama secara eksplisit.

Untuk perubahan kedua:

```
ALTER TABLE guru
  ADD COLUMN aktif SMALLINT UNSIGNED NOT NULL DEFAULT 1
    -- CHECK (aktif=0 OR aktif=1)
  ,
  ADD COLUMN tglkeluar DATE;

ALTER TABLE siswa
  ADD COLUMN aktif SMALLINT UNSIGNED NOT NULL DEFAULT 1
    -- CHECK (aktif=0 OR aktif=1)
  ,
  ADD COLUMN tglkeluar DATE;
```

Kolom *tglkeluar* kita set dapat bernilai NULL. Kita buat konvensi bahwa jika seorang guru/siswa masih aktif, *tglkeluar*-nya diset NULL.

Untuk perubahan ketiga:

```
ALTER TABLE siswa
  ADD COLUMN nama_wali VARCHAR(128) NOT NULL,
  ADD COLUMN hubungan_wali VARCHAR(32) NOT NULL;
```

Dan untuk perubahan keempat:

```
CREATE TABLE tahun (
  tahun SMALLINT PRIMARY KEY
    CHECK (tahun >= 2000 AND tahun <= 2100)
) ENGINE=InnoDB;
INSERT INTO tahun VALUES (2004);

CREATE TABLE tingkat (
  tingkat SMALLINT PRIMARY KEY
    CHECK (tingkat >= 1 AND tingkat <= 6)
) ENGINE=InnoDB;
INSERT INTO tingkat VALUES (1);
INSERT INTO tingkat VALUES (2);
INSERT INTO tingkat VALUES (3);
INSERT INTO tingkat VALUES (4);
INSERT INTO tingkat VALUES (5);
```

```
INSERT INTO tingkat VALUES (6);
```

Kita buat dulu tabel *tahun* dan *tingkat* yang mencatat tahun-tahun ajaran dan tingkatan kelas. Saat ini tidak ada atribut lain yang perlu kita catat mengenai tahun ajaran maupun tingkatan kelas, sehingga kita hanya membuat satu kolom (sekaligus sebagai PK).

Setelah memisahkan tahun dan tingkat, keuntungan yang didapat adalah: kita bisa menjaga agar nanti setiap tahun dan tingkat kelas yang dimasukkan di tabel lain tidak sembarang lagi. Kolom-kolom tahun misalnya, hanya bisa dimasuki tahun 2004. Tahun 2005 baru bisa dimasukkan saat kita telah menambahkan baris 2005 ke tabel *tahun*.

Sekarang yang diperlukan adalah membuat agar tabel-tabel *kurikulum*, *ruangkelas*, *ruangkelas_siswa*, *walikelas*, *nilai*, dan *guru_pelajaran* agar merujuk pada tabel-tabel *tahun* dan *tingkat*:

```
ALTER TABLE kurikulum
  ADD INDEX (tahun),
  ADD FOREIGN KEY (tahun) REFERENCES tahun(tahun),
  ADD INDEX (tingkat),
  ADD FOREIGN KEY (tingkat) REFERENCES tingkat(tingkat);

ALTER TABLE ruangkelas
  ADD INDEX (tingkat),
  ADD FOREIGN KEY (tingkat) REFERENCES tingkat(tingkat);

ALTER TABLE ruangkelas_siswa
  ADD INDEX (tahun),
  ADD FOREIGN KEY (tahun) REFERENCES tahun(tahun);

ALTER TABLE walikelas
  ADD INDEX (tahun),
  ADD FOREIGN KEY (tahun) REFERENCES tahun(tahun);

ALTER TABLE nilai
  ADD INDEX (tahun),
  ADD FOREIGN KEY (tahun) REFERENCES tahun(tahun);

ALTER TABLE guru_pelajaran
  ADD INDEX (tahun),
  ADD FOREIGN KEY (tahun) REFERENCES tahun(tahun),
```

```
  ADD INDEX (tingkat),
  ADD FOREIGN KEY (tingkat) REFERENCES tingkat(tingkat);
```

Sehingga struktur akhir database menjadi seperti di bawah ini (dan hubungan antartabel menjadi seperti pada Gambar 4-8-1).

```
CREATE TABLE tahun (
  tahun SMALLINT PRIMARY KEY
    CHECK (tahun >= 2000 AND tahun <= 2100)
) ENGINE=InnoDB;
INSERT INTO tahun VALUES (2004);

CREATE TABLE tingkat (
  tingkat SMALLINT PRIMARY KEY
    CHECK (tingkat >= 1 AND tingkat <= 6)
) ENGINE=InnoDB;
INSERT INTO tingkat VALUES (1);
INSERT INTO tingkat VALUES (2);
INSERT INTO tingkat VALUES (3);
INSERT INTO tingkat VALUES (4);
INSERT INTO tingkat VALUES (5);
INSERT INTO tingkat VALUES (6);

CREATE TABLE siswa (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nama VARCHAR(128) NOT NULL,
  kelamin CHAR(1) NOT NULL CHECK (kelamin='L' OR kelamin='P'),
  tempatlahir VARCHAR(64) NOT NULL,
  tgllahir DATE NOT NULL,
  tgldaftar DATE NOT NULL,
  aktif SMALLINT UNSIGNED NOT NULL
    DEFAULT 1
    CHECK (aktif=0 OR aktif=1),
  tglkeluar DATE,
  nama_wali VARCHAR(128) NOT NULL,
  hubungan_wali VARCHAR(32) NOT NULL
) ENGINE=InnoDB;

CREATE TABLE ruangkelas (
  nama CHAR(2) PRIMARY KEY CHECK (nama REGEXP '^[1-6][A-D]$'),
  tingkat SMALLINT NOT NULL,
```

```
  INDEX (tingkat),
  FOREIGN KEY (tingkat) REFERENCES tingkat(tingkat)
) ENGINE=InnoDB;

CREATE TABLE ruangkelas_siswa (
  tahun SMALLINT NOT NULL,
    INDEX (tahun),
    FOREIGN KEY (tahun) REFERENCES tahun(tahun),
  id_siswa INT NOT NULL,
    INDEX (id_siswa),
    FOREIGN KEY (id_siswa) REFERENCES siswa(id),
  ruangkelas CHAR(2) NOT NULL,
    INDEX (ruangkelas),
    FOREIGN KEY (ruangkelas) REFERENCES ruangkelas(nama),
  UNIQUE (tahun, id_siswa, ruangkelas)
) ENGINE=InnoDB;

CREATE TABLE guru (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nama VARCHAR(128) NOT NULL,
  kelamin CHAR(1) NOT NULL CHECK (kelamin='L' OR kelamin='P'),
  tempatlahir VARCHAR(64) NOT NULL,
  tgllahir DATE NOT NULL,
  tgldaftar DATE NOT NULL,
  aktif SMALLINT UNSIGNED NOT NULL DEFAULT 1
    CHECK (aktif=0 OR aktif=1),
  tglkeluar DATE
) ENGINE=InnoDB;

CREATE TABLE pelajaran (
  kode CHAR(3) PRIMARY KEY,
  nama VARCHAR(128) NOT NULL
) ENGINE=InnoDB;

CREATE TABLE kurikulum (
  tahun SMALLINT NOT NULL,
    INDEX (tahun),
    FOREIGN KEY (tahun) REFERENCES tahun(tahun),
  kodepelajaran CHAR(3) NOT NULL,
    INDEX (kodepelajaran),
    FOREIGN KEY (kodepelajaran) REFERENCES pelajaran(kode),
```

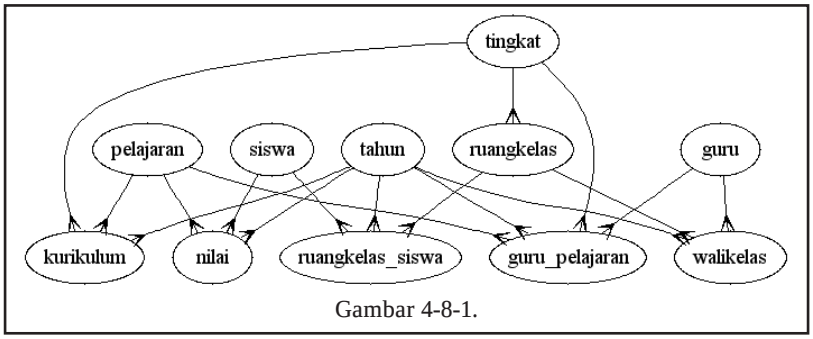
```
  tingkat SMALLINT NOT NULL,
    INDEX (tingkat),
    FOREIGN KEY (tingkat) REFERENCES tingkat(tingkat),
  UNIQUE (tahun, kodepelajaran, tingkat)
) ENGINE=InnoDB;

CREATE TABLE guru_pelajaran (
  tahun SMALLINT NOT NULL,
    INDEX (tahun),
    FOREIGN KEY (tahun) REFERENCES tahun(tahun),
  id_guru INT NOT NULL,
    INDEX (id_guru),
    FOREIGN KEY (id_guru) REFERENCES guru(id),
  kodepelajaran CHAR(3) NOT NULL,
    INDEX (kodepelajaran),
    FOREIGN KEY (kodepelajaran) REFERENCES pelajaran(kode),
  tingkat SMALLINT NOT NULL,
    INDEX (tingkat),
    FOREIGN KEY (tingkat) REFERENCES tingkat(tingkat),
  UNIQUE (tahun, kodepelajaran, tingkat)
) ENGINE=InnoDB;

CREATE TABLE walikelas (
  tahun SMALLINT NOT NULL,
    INDEX (tahun),
    FOREIGN KEY (tahun) REFERENCES tahun(tahun),
  id_guru INT NOT NULL,
    INDEX (id_guru),
    FOREIGN KEY (id_guru) REFERENCES guru(id),
  ruangkelas CHAR(2) NOT NULL
    CHECK (tingkat >= 1 AND tingkat <= 6),
  UNIQUE (tahun, ruangkelas)
) ENGINE=InnoDB;

CREATE TABLE nilai (
  tahun SMALLINT NOT NULL,
    INDEX (tahun),
    FOREIGN KEY (tahun) REFERENCES tahun(tahun),
  caturwulan SMALLINT NOT NULL
    CHECK (caturwulan >= 1 AND caturwulan <= 3),
  id_siswa INT NOT NULL,
```

```
    INDEX (id_siswa),
    FOREIGN KEY (id_siswa) REFERENCES siswa(id),
  kodepelajaran CHAR(3) NOT NULL,
    INDEX (kodepelajaran),
    FOREIGN KEY (kodepelajaran) REFERENCES pelajaran(kode),
  nilai SMALLINT
    CHECK (nilai >= 0 AND nilai <= 10),
  UNIQUE (tahun, caturwulan, id_siswa, kodepelajaran)
) ENGINE=InnoDB;
```



Gambar 4-8-1.

4.8 Contoh Latihan ALTER TABLE